**COSC 352.001 – Organization of Programming Languages**

**Fall / 2017**

**Project 3**

**Name: Jose Dixon**

**Instructor: Vojislav Stojkovic**

**Points:**

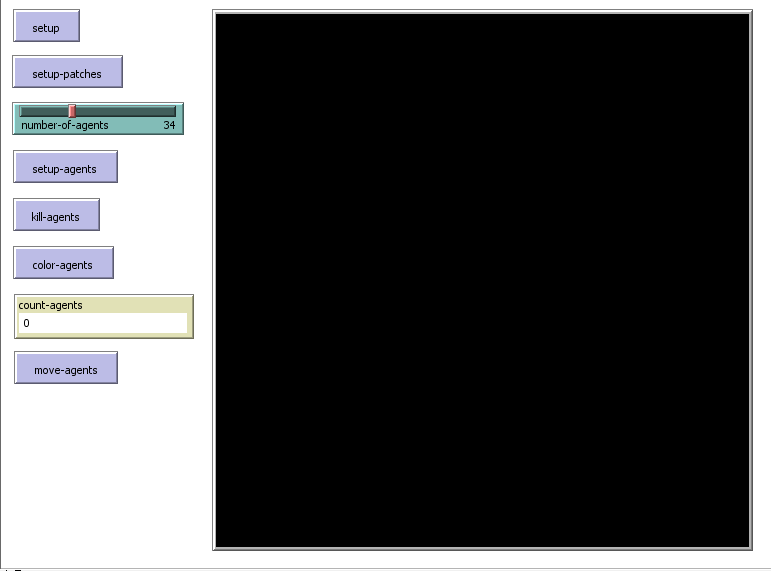
**Part II. Description of Text/Problem**

# Problem

Write a NetLogo program to:

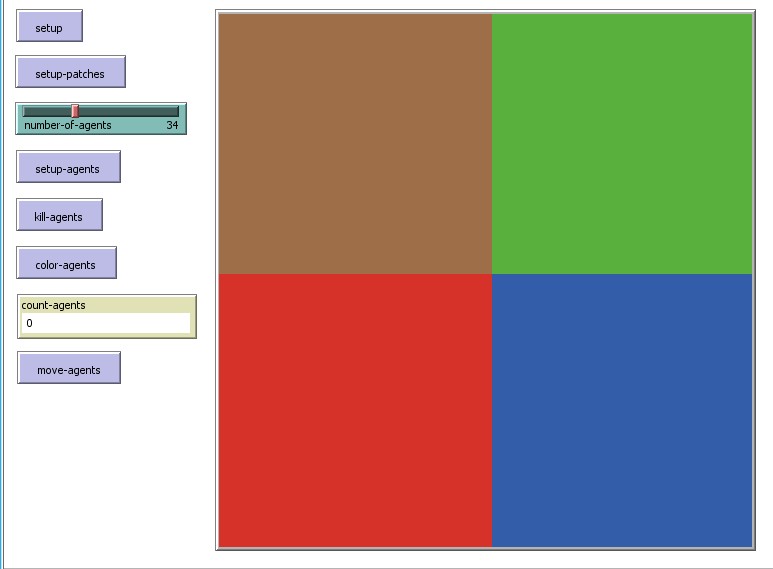
**(a) 10 points**

Set up initial environment.



**(b) 20 points**

Set up patches in four colors.



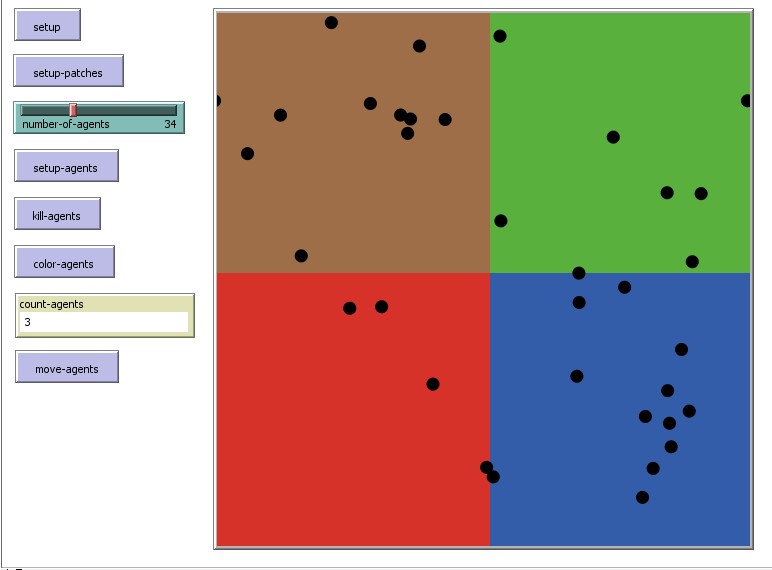
**(c) 10 points**

Set up agents.

Number of agents is defined by slider.

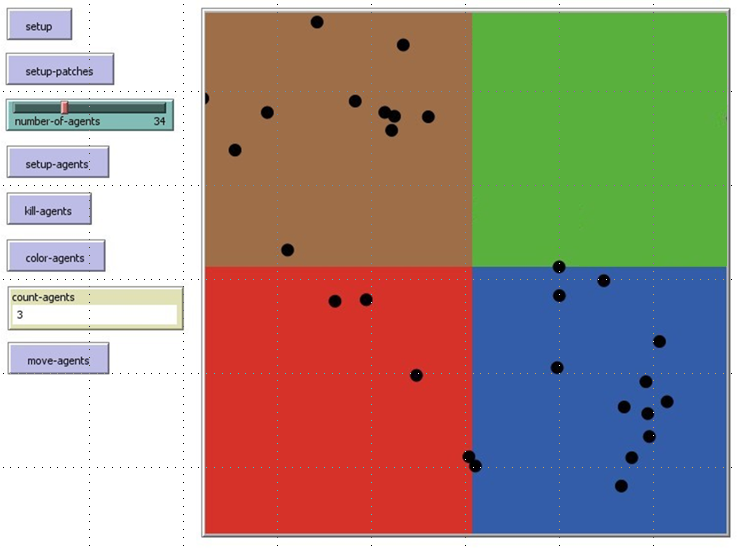
Positions of agents are random.

Shape of agents is circle.



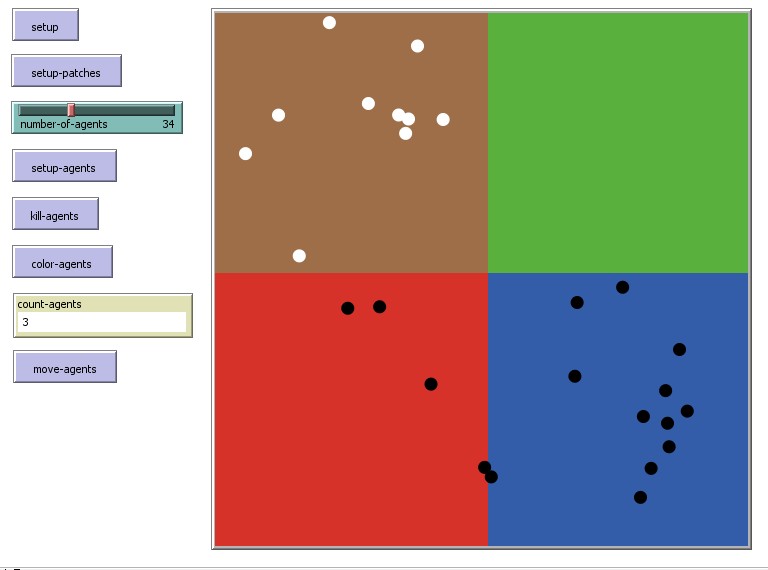
**(d) 15 points**

Kill agents in the first quadrant.



**(e) 15 points**

Color agents in the second quadrant in white.



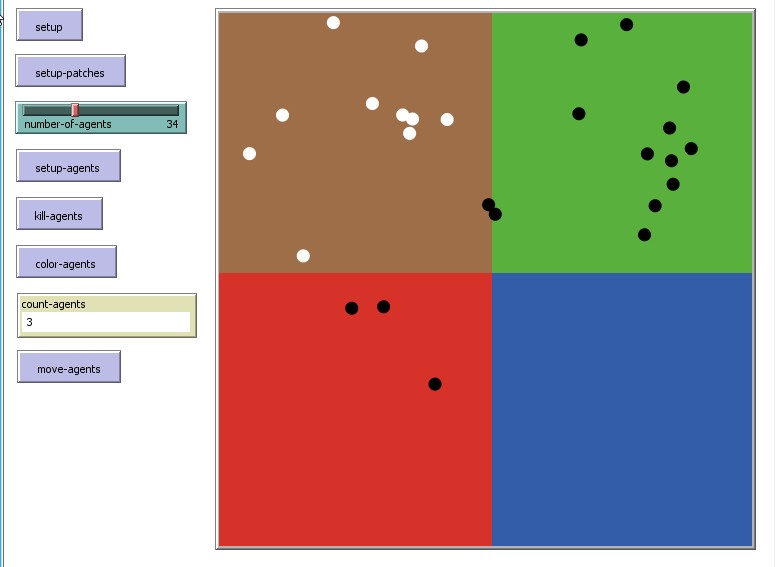
**(f) 15 points**

Count agents in the third quadrant

**4**

**(g) 15 points**

Move agents from the fourth quadrant into the first quadrant.



**Part III. Page 4 – Algorithms/Data Structures and Explanations**

* You have four global variables specified for the four quadrants.
* The program starts with setup that clears everything.
* Setup set each quadrant into a patch with its own color by using random-pxcor and pycor keywords.
* Setup agents uses circle turtles that are black and randomly located in the program.
* Kill agents determines if the patch is green it would delete any agents in the green patch using die keyword..
* Color agents determines if the patch is orange to change any agents from black to white.
* Count agents counts the agents in the third quadrant if they are contained in the blue patch.
* Move agents moves any agents if the patch color is red from the fourth quadrant to the green quadrant.

**Page 5 – Program Code**

(define Graph

'((A (B E))

(B (E F))

(C (D))

(D ())

(E (C F))

(F (D G))

(G ())))

(define (make-graph graph)

(cond [(empty? graph)]

[(cons? graph) ((first (first graph))

(second (first graph))

(make-graph (rest graph)))]))

(define (neighbors a-node a-graph)

(cond [(empty? a-graph) (error "Vertex not in graph")]

[(symbol=? a-node (first (first a-graph))) (second (first a-graph))]

[else (neighbors a-node (rest a-graph))]))

(define (find-route origination destination graph)

(cond [(symbol=? origination destination) (list origination)]

[else (local [(define nbrs (neighbors origination graph))

(define route (find-route/list nbrs destination graph))]

(cond [(false? route) route]

[else (cons origination route)]))]))

(define (find-route/list los destination graph)

(cond [(empty? los) false]

[else (local [(define route (find-route (first los) destination graph))]

(cond [(false? route) (find-route/list (rest los) destination graph)]

[else route]))]))

**Page 6 – Test Examples**

(find-route 'B 'G Graph) -> ‘(B E FG)

(find-route ‘A ‘G Graph) -> ‘(A C F B G)